Universitat de Lleida

TREBALL FINAL DE GRAU

Estudiant: Joan Pau Castells Gasia

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: A scalable parallel Progressive Hedging Algorithm for stochastic cluster-scenario-based mixed-integer models

Director/a: **Jordi Mateo Fornés i Adela Pagès Bernaus**

Presentació

Mes: Juliol

Any: 2019

A scalable parallel Progressive Hedging Algorithm for stochastic cluster-scenario-based mixed-integer models

Author: Joan Pau Castells

July 10, 2019

Contents

List of Figures

List of Tables

Abstract

This work presents a general parallelisation of the Progressive Hedging algorithm to coordinate the resolution of two-stage and multi-stage stochastic mixed-integer problems without (binary or integer) variables in the first stage. We report a benchmark study between the computational improvements using our proposal and the parallel version (using pyro) of the Pyomo integrated Progressive Hedging. Moreover, we study the influence of a quadratic term to accelerate the convergence, different scenario-cluster formation and several step update policies by solving different instances using our proposal.

Keywords Progressive Hedging Algorithm ; stochastic mixed-integer optimization ; parallelization; scalability; clustering in stochastic optimisation

Chapter 1

Introduction

Many real-world problems can be modelled using mathematical programs to optimise (minimise or maximise) an objective function taking into account a set of constraints which the optimal solution has to satisfy. The decisions are represented by variables and the constraints are the limits, minimum requirements of the problem[1]. Decision-makers used to assume the future was determined to decrease the complexity of the problems. But real-problems are very difficult and uncertain to predict. So, tools are needed by decision-makers to study the behaviour of uncertain parameters related to their problems.

Stochastic problems are top-rated tools to deal with uncertainty. They are mathematical programs where some of the data that is uncertain affects the objective function, the parameters and also the constraints. Some applications of stochastic programming are energy[2], logistics[3],healthcare[4],finance[5], among others.

In the context of two-stage stochastic problems the decisions are taken in two stages. The decisions that have to be taken without knowing about the information of uncertain parameters are in the first stage, and in the second stage there are the decisions where full uncertain information is known and represent the corrective actions of the first-stage decisions.

In the context of multi-stage stochastic problems there are more than two stages, at every stage the uncertain information is being revealed, so the decisions taken in one stage can be modified in the next stage with its corrective actions.

This paper focuses on two-stage and multi-stage problems in which the uncertainty of the parameters is represented by discrete and finite distributions for mixed-integer programs.

There are effective techniques to solve two-stage problems such as the integer L-shaped method [6], Lagrangian Decomposition [7], Progressive Hedging [8], among others. The Progressive Hedging algorithm proposed by Rockafellar and Wets is a scenario-based decomposition technique that can be used to solve such problems. It was originally used with problems containing only continuous variables. Referred to as a horizontal decomposition since decomposes stochastic

programs by scenarios rather than by time-stages. It can also be used as a part of more complex algorithms for example the BFC(Branch and Fix Coordination) [9] which is based on the same scenario decomposition scheme.

The improvement of the Progressive Hedging algorithm needs to be investigated since critical issues appear with large-scale mixed-integer problems that can result in either non-convergence or extremely long run-times. Authors in [8] present innovation techniques in order to improve the efficiency of the Progressive Hedging for a two-stage stochastic network flow problem with integer variables in both stages.

We used the update policies described in [8]. We propose a new policy to try to improve one of these policies to reduce the number of iterations.

Standard mixed-integer programming solvers can be used to solve, the extensive form of the problem in which all the scenarios are considered simultaneously, small and easy problems, but generally due to the size of the problems and the difficulty, they have issues or are unable to solve them. So the best way to deal with this kind of problems is using decomposition techniques.

For very large instances it may be that there is not enough memory to build the full model within one computer. To deal this issue we use a decomposition method called Lagrangean Decomposition [7] so the full model is not needed anymore.

Our main contribution is the implementation of a parallelization of the Progressive Hedging to solve large-scale problems that standard mixed-integer programming solvers can not solve in a reasonable time and improve the performance in comparison to the current commercial implementations like Pyomo Integrated Progressive Hedging.

We use a set of instances to compare the performance of our implementation with the Pyomo integrated Progressive Hedging and a standard commercial solver called CPLEX; to check the behaviour of the different update policies and if there is an improvement of the performance depending on the choice of the policy; to study how affects to the performance the addition of a quadratic term to the objective function and finally how the scenario-cluster decomposition affects to the convergence and the performance of our proposal.

Chapter 2

Preliminaries

This section presents the theoretical background of the algorithm implemented. First the general Progressive Hedging algorithm is presented and follows the extension where scenarios are clustered. The algorithm is presented for a two-stage model with continuous variables. The same ideas can be applied to multi-stage models. The implementation accepts multi-stage models with binary variables in some parts.

2.1 Progressive Hedging Algorithm

Consider a general two stage stochastic mixed-integer programming model (2.1):

$$
\begin{aligned}
(EF) \qquad & \min c'x \ + \Xi[f(x,\tilde{\xi})] && \text{(2.1a)}\\
s.t.: \quad & Ax \geq b && \text{(2.1b)}\\
& x \in \mathbb{R}^{n1} && \text{(2.1c)}
\end{aligned}
$$

The random vector $\tilde{\xi}$ is defined on a probability space (Ω, A, P). Given a particular realisation ξ of $\tilde{\xi}$, the recourse function $f(x,\tilde{\xi})$ is defined as:

$$
\begin{aligned}
& f(x,\xi) = \min \ g(\xi)'y(\xi) && \text{(2.2a)}\\
s.t.: \quad & Wy(\xi) \geq r(\xi) - Tx && \text{(2.2b)}\\
& y(\xi) \in \mathbb{R}^{n2} && \text{(2.2c)}
\end{aligned}
$$

The random vector $\tilde{\xi}$ is represented by a set of scenarios and has a finite support in Ω and a representative set of realisations with corresponding probabilities p_ξ are available. Assuming this, the stochastic program expressed in Model (2.1) can be expressed as a weighted sum and be rewritten as a Deterministic Equivalent Model (DEM) in compact formulation as in Model (2.3).

$$
\begin{aligned}
(DEM) \qquad & \min\ c'x + \sum_{\xi \in \Omega} p_\xi g(\xi)' y(\xi) && && (2.3a)\\
s.t.: \quad & Ax \geq b && && (2.3b)\\
& Wy(\xi) \geq r(\xi) - T(\xi)x && \xi \in \Omega && (2.3c)\\
& x \in \mathbb{R}^{n1} && && (2.3d)\\
& y(\xi) \in \mathbb{R}^{n2} && \xi \in \Omega && (2.3e)
\end{aligned}
$$

The non-anticipativity constraints (NAC) are satisfied implicitly. These constraints are necessary to guarantee that decisions will not depend on the future results of a random event, but will do on available information until the decision moment. A formulation which expresses explicitly the NAC, uses a variable for each scenario and enforces these constraints in a explicit manner.

$$
\begin{aligned}
& \min \sum_{\xi \in \Omega} p_\xi [c'x(\xi) + g(\xi)' y(\xi)] && && (2.4a)\\
s.t.: \quad & Ax \geq b && \xi \in \Omega && (2.4b)\\
& Wy(\xi) \geq r(\xi) - Tx && \xi \in \Omega && (2.4c)\\
& p_\xi x(\xi) - p_\xi \hat{x} = 0 && \xi \in \Omega && (2.4d)\\
& \hat{x}, x(\xi) \in \mathbb{R}^{n1} && \xi \in \Omega && (2.4e)\\
& y(\xi) \in \mathbb{R}^{n2} && \xi \in \Omega && (2.4f)
\end{aligned}
$$

While the compact formulation uses less variables and constraints, the structure of a scenario-based formulation can be exploited in a solution method based on decomposition.

2.2 Progressive Hedging Cluster Decomposition

A mixed formulation with the compact and scenario-based formulation can be used, in a way where some scenarios are clustered.

Assume that the set of scenarios is split into $|K|$ disjoints clusters of n_k scenarios. Each cluster Ω_k may have a different number of scenarios, but the union of all clusters is equal to the original set of scenarios. The probabilities

of each cluster k is $p_k = \sum_{\xi \in \Omega_k} p_\xi$. This formulation can be expressed as:

$$\min \sum_{k \in K} p_k (c' x(k) + \sum_{\xi \in \Omega_k} \frac{p_\xi}{p_k} g(\xi)' y(\xi)) \tag{2.5a}$$

$$s.t.: \quad Ax(k) \geq b \qquad k \in K \tag{2.5b}$$

$$Wy(\xi) \geq r(\xi) - Tx(k) \qquad \xi \in \Omega \tag{2.5c}$$

$$p_k x(k) - p_k \hat{x} = 0 \qquad k \in K \tag{2.5d}$$

$$\hat{x}, x(k) \in \mathbb{R}^{n1} \qquad k \in K \tag{2.5e}$$

$$y(\xi) \in \mathbb{R}^{n2} \qquad \xi \in \Omega \tag{2.5f}$$

This is a general mixed formulation since for one cluster this is the compact formulation (2.3) and for a partition with the number of scenarios this is the scenario-formulation (2.4).

To illustrate the ideas behind the different formulations, a small example is presented. The stochastic parameters and their dependencies can be represented in form of a tree. For example, in Figure 2.1 node $N0$ represents the information on the stochastic parameters at the first stage. There is also associated a set of decisions that can be taken based on this information. Children nodes represent new information with respect to the stochastic parameters. In this example, four different values may occur. This leads to four different scenarios, $\Omega = \{1, 2, 3, 4\}$. A scenario is the path from the root node to a leaf node and corresponds to one realization of the whole set of the uncertain parameters.

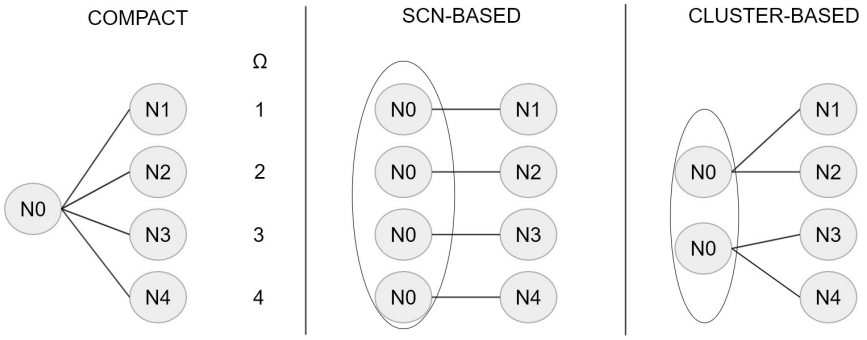


Figure 2.1: Different representations of the scenario tree

The mathematical model which uses one variable per node is the most compact formulation, in terms of number of variables. An alternative formulation is to use different variables for each scenario, and force to have the same values at those points where the uncertain parameters have the same information. This is represented in Figure 2.1 in the central part (Scn-based), where copies of the root node are made for each scenario and then non-anticipaty constraints are used.

A mixed strategy is to cluster the scenarios in groups and replicate the root node, one node for each cluster and add non-anticipativity constraints to ensure that the same decisions will be taken in each replica. This approach is presented in Figure 2.1 at the right-hand side, where $K = 2$, $\Omega_1 = \{1, 2\}$ and $\Omega_2 = \{3, 4\}$.

Chapter 3

Pseudo-code

This pseudo-code is based on the one presented in [8] and uses the cluster decomposition formulation.

A penalty factor $\rho > 0$ and a termination threshold ϵ are given to the algorithm. The pseudo-code of the algorithm is presented in Algorithm (1). The Progressive Hedging Algorithm exploits the fact that relaxing the non-anticipativity constraints results in a collection of much smaller subproblems that can be solved faster. The algorithm handles the NACs by adding them in the objective function penalized with a constant.

When relaxing the NACs of model (2.5), k subproblems as model (3.1) appear and are solved many times.

$$
\begin{aligned}
SP_k(X,Y): \quad & \min\ p_k(c'x(k) + \sum_{\xi\in\Omega_k} \frac{p_\xi}{p_k} g(\xi)'y(\xi)) && && \text{(3.1a)}\\
s.t.: \quad & Ax(k) \geq b && && \text{(3.1b)}\\
& Wy(\xi) \geq r(\xi) - Tx(k) && \xi \in \Omega_k && \text{(3.1c)}\\
& \hat{x}, x(k) \in \mathbb{Z}_+^{m1} \times \mathbb{R}^{n1-m1} && && \text{(3.1d)}\\
& y(\xi) \in \mathbb{Z}_+^{m2} \times \mathbb{R}^{n2-m2} && \xi \in \Omega_k && \text{(3.1e)}
\end{aligned}
$$

When the first iteration of the algorithm ($ite = 0$) completes then, in the next iterations, it changes the coefficients of the coordinated continuous variables in the objective function according to Equation(3.2).

$$
f^k(x,y) = p_k c'x(k) + \sum_{\xi\in\Omega_k} p_\xi (g(\xi)'y(\xi) + s_k^{(ite-1)} x(k) + \frac{\rho}{2} ||x(k) - \overline{X}^{(ite-1)}||^2) \tag{3.2a}
$$

This function account for an Augmented Lagrangian term which is multiplied by the ρ parameter. Vector s_k is the step direction which is computed at each

iteration of the algorithm, and $\overline{X}^{(ite)}$ is the average value of the coordinated variables (NAC) at each iteration.

Algorithm 1 Progressive Hedging

$ite = 0$
$\forall\ k \in K, (X_k^{ite}, Y_k^{ite}) \leftarrow$ solve subproblem $\mathrm{SP}_k(X, Y)$
Compute the average solution $\overline{X}^{ite} = \sum_{k \in K} p_k X_k^{ite}$.
Compute the step: $\forall\ k \in K, s_k^{ite} = \rho(X_k^{ite} - \overline{X}_k^{ite})$
Compute the error $e^{ite} = \sum_{k \in \Omega_k} p_k \mid\mid X_k^{ite} - \overline{X}^{ite} \mid\mid$
Update ρ with its policy and value
Set isCompleted= false
while !isCompleted **do**
    ite++
    $\forall\ k \in K$, solve $\mathrm{SP}_k(X, Y)$ with objective function (3.2)
    Compute the average value $\overline{X}^{ite} = \sum_{k \in K} p_k X_k^{ite}$
    Compute the step: $\forall\ k \in K, s_k^{ite} = s_k^{ite-1} + \rho(X_k^{ite} - \overline{X}_k^{ite})$
    Compute the error $e^{ite} = \sum_{k \in K} p_k \mid\mid X_k^{ite} - \overline{X}^{ite} \mid\mid$
    if $e^{ite} \leq \epsilon$ **then** isCompleted=True
    end if
end while

Chapter 4

Implementation

This implementation is coded in C++ since it is one of the fastest programming languages, provides an excellent concurrency support, which plays an important role in terms of performance, and it has a lot of control on how the implementation uses the resources [10]. It uses the MPI(Message Passing Interface) protocol [11] to communicate data between processes.

The Progressive Hedging can be used in sequential to solve a problem, but it has an adequate structure to be solved in parallel because of the independence of the subproblems. Our implementation uses the cluster decomposition which is done in the design stage of the model. Each subproblem is solved in a particular processor using CPLEX to solve it.

4.1 Addition of a quadratic term to the objective function

In the practise arise critical problems when implementing Progressive Hedging for particular cases especially in the context of very difficult or large-scale mixed integer problems. Failure to address these issues results in either non-convergence or unreasonable long run times. One way to reduce these issues is adding a quadratic term to the objective function which speeds up the algorithm and converges in a reasonable time. Objective function (3.2) includes the quadratic term $\frac{\rho}{2}||x(k) - \overline{X}^{(ite-1)}||^2)$.

If we do not add this quadratic term in each iteration the coefficients will be changed by the linear function and it will produce either long run-times or non-convergence. Using the quadratic objective function (3.2) the new objective coefficients of the coordinated continuous variables are computed in this way.

$$f^k(X_k^{ite}) = \frac{\rho}{2}(X_k^{ite})^2 + X_k^{ite}(c + s_g^{(ite-1)} - \rho\overline{X}^{(ite-1)}) + \frac{\rho}{2}\overline{X}^{(ite-1)} \tag{4.1a}$$

In (4.1) the quadratic term is multiplied by $\frac{\rho}{2}(X_k^{ite})^2$, the linear part is $X_k^{ite}(c + s_g^{(ite-1)} - \rho\overline{X}^{(ite-1)})$ and $\frac{\rho}{2}\overline{X}^{(ite-1)}$ is an independent term. This function is

obtained by converting the quadratic term $(a+b)^2$ to (a^2+b^2+2ab) and sum each term with each equal term of the equation.

4.2 Update of the penalty factor

Progressive Hedging is an algorithm which is very sensitive to the arguments it receives. The choice of the penalty factor ρ helps converge in a reasonable time. There are situations where if the penalty factor ρ is too slow then the algorithm will require many iterations to converge. For this reason in this implementation we have tested different update policies to help the algorithm reach faster the optimal value. Some of the policies are published in [8]. The update policies are:

1. FX(.): This policy receives a parameter which represents a fixed value for the penalty factor for all the variables in all iterations.

2. CP(.): The penalty factor of this policy for a specific variable i is the multiplier ($m > 0$) received, multiplied by the cost of the variable i.

3. SEP(): It computes the penalty factor dividing the cost of a specific variable and the maximum of 1 and the iteration error e^0 (See algorithm 1). $\rho(i) = \frac{c(i)}{max(e^{ite},1)}$. The advantage of this policy is that it is parameter-free, so there are not high-quality search ρ, and are not needed repeated executions of the algorithm to search its optimal value.

4. SEP_IT: This policy is an improvement of the SEP policy which is a new proposal of the thesis. A multiplication factor α is introduced to the formula together with a lower bound and an upper bound (see Equation 4.2). This mechanism tries to avoid iterations in which the improvement of the objective function has insignificant improvement or an excessive improvement, so this α will be updated depending on a ratio and these bounds. This α is equal to 1 if $ite = 0$, but in next iterations this multiplication factor is updated taking into account the ratio. This ratio is obtained at iteration $ite\%3 == 0$, it is updated with the sum of the objective function at this iterations as (f(ite-1)-f(ite-2))+(f(ite) - f(ite-1)). This ratio represents how it has changed the objective function in these 3 iterations. The multiplication factor is multiplied by 2 if this ratio is lower than the lower bound and multiplied by 0.5 if this ratio is greater than the upper bound.(See algorithm 2 where the first if statement represents the step 6 and the else statement the following step after step 13 of the algorithm 1).

$$\rho(i) = \alpha * \frac{c(i)}{max(e^{ite}, 1)}. \tag{4.2a}$$

Algorithm 2 SEP_IT update policy

```
1: α = 0
2: if ite=0 then
3:     α = 1
4:     values = []
5:     Update ρ values with equation (4.2)
6: else
7:     Set objective value to position (ite-1)%3 of values array.
8:     if ite%3=0 then
9:         ratio=(values[1]-values[0])+(values[2]-values[1])
10:        if ratio < lowerbound then
11:            α = α * 2
12:        else if ratio > upperbound then
13:            α = α * 0.5
14:        end if
15:        Update ρ values with equation (4.2)
16:    end if
17: end if
```

4.3 Parallelization of the Progressive Hedging

When solving a difficult or large-scale mixed integer problem, the Progressive Hedging can require a lot of time to solve it. To reduce the computational time we propose a cluster parallel algorithm where scenarios are distributed between the number of clusters chosen to solve the problem. The entire problem is decomposed in k subproblems, in which k represents the number of clusters (subproblems). These subproblems can have an equal size or a different size, where this size depends on the number of scenarios it contains.

This implementation is based on a master/ worker model. This pattern is adequate for this algorithm since the subproblems have not dependency between them. So these subproblems can be solved between the different workers without knowing of the existence of the others.

The parallelization is made by the Message Passing Interface protocol, the master and workers have their private memory and the synchronisation is made by communicating data by messages[11]. The task of the master process is to compute new objective coefficents for workers, wait for the subproblems solution, compute the average solution of the workers solutions, compute the step direction and compute an error to decide when to stop. On the other hand the workers responsibility is to solve their subproblems and send the solutions to the master processor.

The main workload here is to solve all the subproblems. Since the subproblems are independent between them they can be designated to the workers which represent processors. The more processors designated are, the more small the subproblem becomes, but the more the master processor has to wait for the compute of the workers. When using an MPI approach, developers need to

consider the trade-off between the cost of preparing the workers, the communication hotspots, and the granularity (size of the subproblem to be solved by each worker). This way, the subproblems have to be as the same order of complexity to exploit the master-worker paradigm.

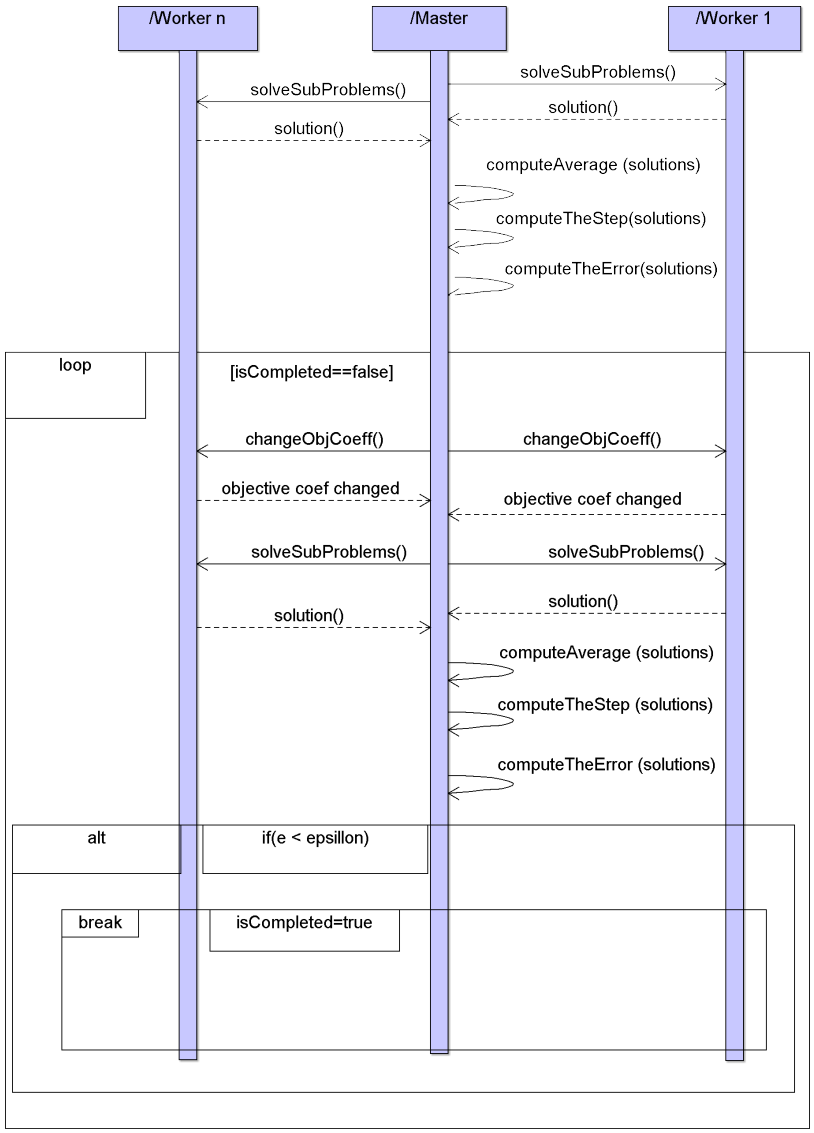


Figure 4.1: Master/Worker Progressive Hedging

The algorithm works in this way, in the first iteration the master tells the workers to solve their subproblems with the original objective coefficients. The workers solve the subproblems and send them to the master, meanwhile the master waits for the solutions. Once the master receives the solutions from the workers, it can proceed to compute the average solution, compute the step and compute the iteration error and finally updates the ρ depending on the update policy. Once the first iteration is completed, the following tasks are repeated until the iteration error is minor to a ϵ: a new task is added to the master processor, it has to compute the new objective coefficients depending on the objective function which has been chosen to run the Progressive Hedging which are linear or quadratic (equation 3.2) and send them to the workers. When the workers receive the new objective coefficients they update the coefficients of their objective function. When the master is notified by all the workers they have changed the objective coefficients, then they continue with the same tasks as the first iteration.

Figure 4.2 shows a task dependency graph which shows the order tasks are executed and figure 4.1 shows a sequence diagram which shows the communication between the master and workers. When solving the subproblems the master processor waits for the solution of workers to continue with its work. (See figure 4.2). This functionality can prevent the algorithm from running as fast as it should. This phenomenon is called as a bottleneck [11].

Figure 4.1 and 4.2 highlight the following:

1. Performance bottleneck in this code is the data dependency between task solveSubProblems and task computeTheAverage.

2. The main hotspots in the implementation can be shown in the master-worker communication when exchanging data related to step solveSub-Problems

3. Load balancing could be a problem when dealing with a heterogeneous scenario cluster, but not in the homogenous case.

The larger (in terms of number of scenarios) a problem is the more processors can be used to solve the subproblems that represents the entire problem to solve it in a reasonable time, and the size of the subproblem for a specific processor decrease. By contrast the less processors are used to solve the problem, the more scenarios have the subproblems.

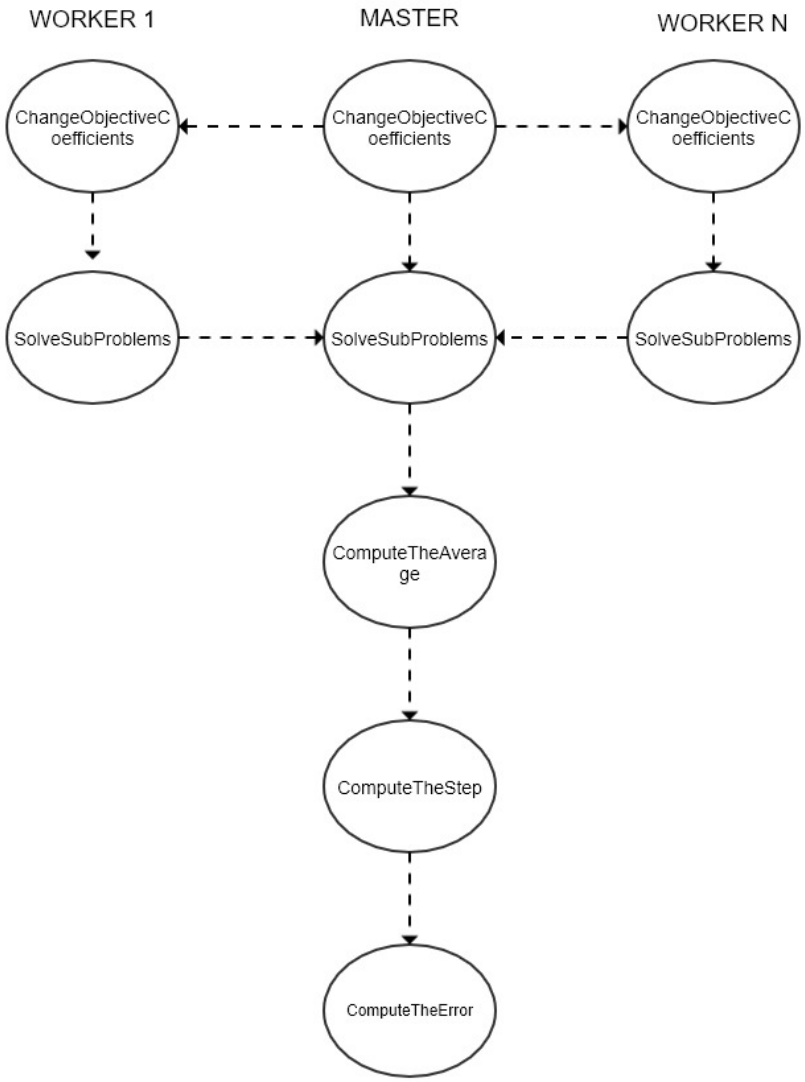


Figure 4.2: Task dependency graph

Chapter 5

Test Study

Tests were conducted in a cluster simulated inside an OpenNebula (a cloud infrastructure) [12] with 2,2 Ghz, 20 GB of RAM, 60 CPU with 1 core and 1 physical thread in each core with CentOS 7 as the operating system. Some instances of the test-bed were from a Python version 3.6 program which uses an open source package called Pyomo [13] to formulate, solve and analyse optimisation models.

The models used in this program are based on the farmer example proposed by Birge and Louveaux [14], a farmer who wants to decide how many crops to plant (1st stage variables) and then buying or selling this crops taking into account some requirements about the amount of each crop it has reached.[15].

The scenarios of the farmer problems represent the different yields for the crops the farmer has experienced over the years. In the instance F1 (see Table 5.1) these scenarios represent the yields for the crops in good, average and bad weather conditions varying the average a 20% to positive and negative.

We generate new instances (from F2 to F10 see Table 5.1) increasing the size of the problem (in terms of scenarios) where the new scenarios were distributed between bad and good scenarios until they reach the original good and bad yields of instance F1. To generate the files the Pyomo Progressive Hedging needs to run this instances which basically are the scenario data (*Scenario.dat) and a file which represents the scenario tree (ScenarioStructure.dat). Moreover, we need to generate the files requirede for the Progressive Hedging, TotalProblem.mps which represents the problem in compact formulation, a file for each cluster called Cluster*.mps which represents the subproblems and a file for each cluster called VariableDictionary.txt which link the variables of the subproblems with the original problem.

Table 5.1 shows the dimensions of the stochastic models for such problems. The main parameters were the following: $|\Omega|$, *number* of scenarios; n_x, number of variables in the first stage; n_y, number of variables in the second stage; nc, number of constraints, nv, number of variables and ns the number of the stages.

The second set, the f00x are multi-stage stochastic programs where we wanted to study how our implementation behaves when solving this kind of

Instance	$\|\Omega\|$	n_x	n_y	nc	nv	ns
F1	3	3	28	37	31	2
F2	15	3	136	181	139	2
F3	30	3	271	361	274	2
F4	45	3	406	541	409	2
F5	60	3	541	721	544	2
F6	75	3	676	901	679	2
F7	90	3	811	1081	814	2
F8	100	3	901	1201	904	2
F9	200	3	1801	2401	1804	2
F10	1000	3	9001	12001	9004	2
f001	36	352	35552	85911	35904	17
f002	36	1760	177.760	429555	179520	17

Table 5.1: Model dimensions. Compact representation

problems. These instance were obtained from a local database and are focused to solve the optimal delivering of fattened pigs to the slaughterhouse.

The objective for using these instances was to compare how better was our proposal compared with a commercial version of the same algorithm and with commercial solvers such as CPLEX. Moreover, we want to highlight how the initial parameters affect the performance: the choice of a rho policy, the choice of a linear or quadratic function and the number of clusters used to solve the problem.

5.1 Pyro-ph vs Progressive Hedging

Pyomo is a popular open-source package to formulate and solve optimisation problems which is used by many government agencies, academic institutions and applications [13]. It contains an implementation of the Progressive Hedging called runph and taking advantage of a python package called Pyro, runph can be run in parallel using an MPI approach. Using remote solvers and sending the subproblems to be dispatched to this parallel solvers.

First of all, we compared the performance of our implementation of the Progressive Hedging with the pyro-ph using the same conditions. Each implementation solved the Farmer instances set with a FX rho policy with a value of 1.0 and using CPLEX to solve the subproblems. Figure 5.1 shows the performance differences between the pyro-ph and our proposal (PH). This figure shows how our implementation reduced the execution time. As it shows the more scenarios

had the problem, the more time pyro-ph needs to solve the problem, however for our implementation this scenario increment slightly affects its performance.

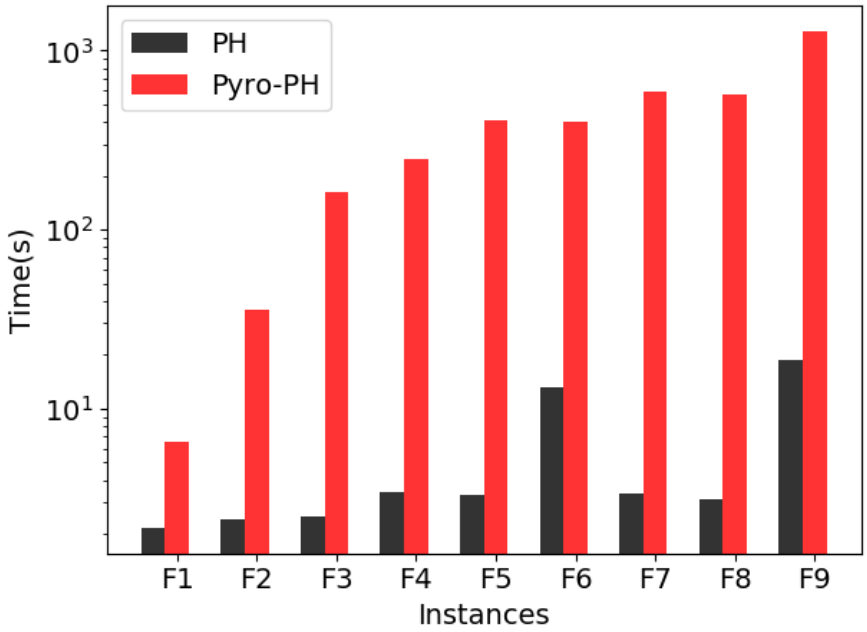


Figure 5.1: Progressive Hedging vs Pyomo Progressive Hedging(time)

Our implementation solved instance F10 in 74,81 seconds, but Pyro-ph was unable to solve it. This way, we can conclude that for the farmer stochastic instances we are able to beat pyro-ph.

5.2 Progressive Hedging vs CPLEX

We compared the difference between CPLEX 12.8 (which includes algorithms to take advantage of the cores, these cores are run in a virtual machine with the same capacities as the Cluster MPI) and our implementation of the Progressive Hedging. Figure 5.2 shows the performance between the effective form solved by CPLEX and the parallelization of the problem solved by Progressive Hedging.

This figure shows how there is no need to use a parallelization for small mixed-integer problems (Farmer instances set) since CPLEX can solve them without problem and Progressive Hedging loss time preparing the parallel environment. By contrast the instances f00x contain enough binary variables to increase the difficulty of the problem to be solved by CPLEX in a way that it becomes to have similar times with our implementation.

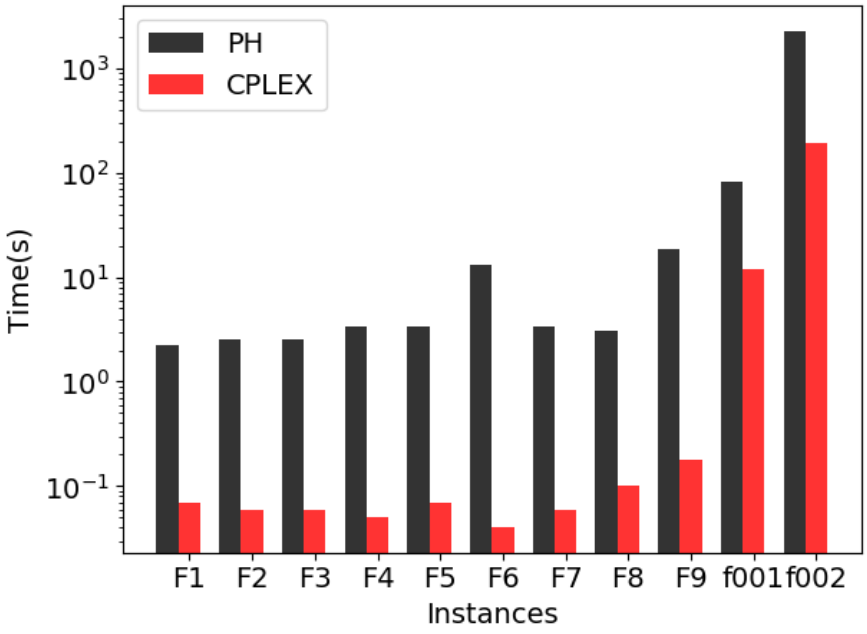


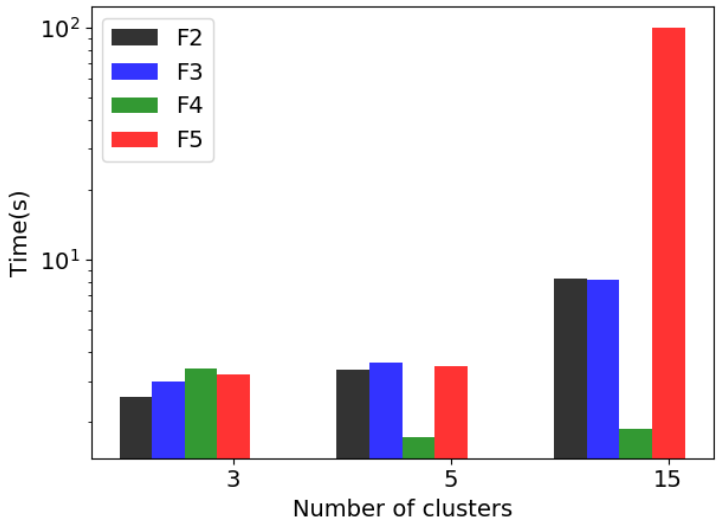
Figure 5.2: Comparison with CPLEX

5.3 Progressive Hedging scenario-cluster formation

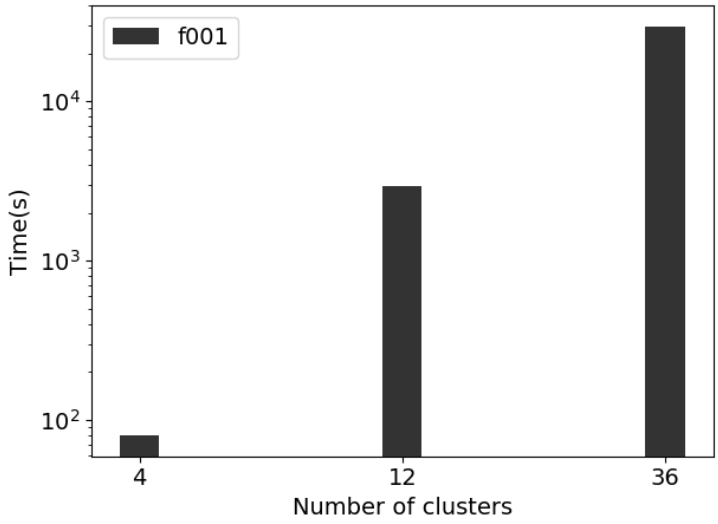
In this section, we studied the behaviour of the Progressive Hedging according to the number of clusters assigned to solve the problem.

Figure 5.3 shows how our implementation found for instance F2 an optimal solution at iteration 147 in 2.56 seconds with k=3 (which represents three processors for the workers + one processor for the master). However, if the number of processors increase seems to also increase the time to find a solution and the number of iterations since for the size of this problem there is no need to split the problem in more subproblems because it is easy to solve in only two clusters and adding more processors only add more time of process initialisation and synchronisation time. If the size of the problem is the bigger enough to add more clusters then a reduction of the time to find a solution is done.

The results in figure 5.3 shows that an efficient performance of our implementation depends on the number of clusters and the size of the problem, since a small problem needs few clusters, and if it is bigger then more clusters are required. In instance f4 using k=3 found the optimal solution in 3,38 seconds and 121 iterations, however increasing the number of clusters to 5 reduced the time the 50% and the number of iterations to 17.



(a) Farmer instances



(b) f001

Figure 5.3: Influence of clusters

By contrast in problem f001, figure 5.3 shows how using k=4 finished at 2 iterations in 79.37 seconds finding a suboptimal. (CPLEX found an optimal solution which was an objective value of 3.8244444080e+05). However increasing the number of cluster to 12 it found the same solution as CPLEX in 49 minutes but increasing the number of cluster to 36 it increased the time exponentially, it found a sub-optimal solution in approximately 8 hours, it seemed it would have found the optimal solution if the ϵ which determines if the solutions of the different scenarios are equal was lower.

5.4 Quadratic vs Linear Objective function

In this section we studied how affects solving the instance F1 (which is the easiest) from the Farmer set problems with a FX rho policy with value 1.0 and using a quadratic objective function or a linear objective function.

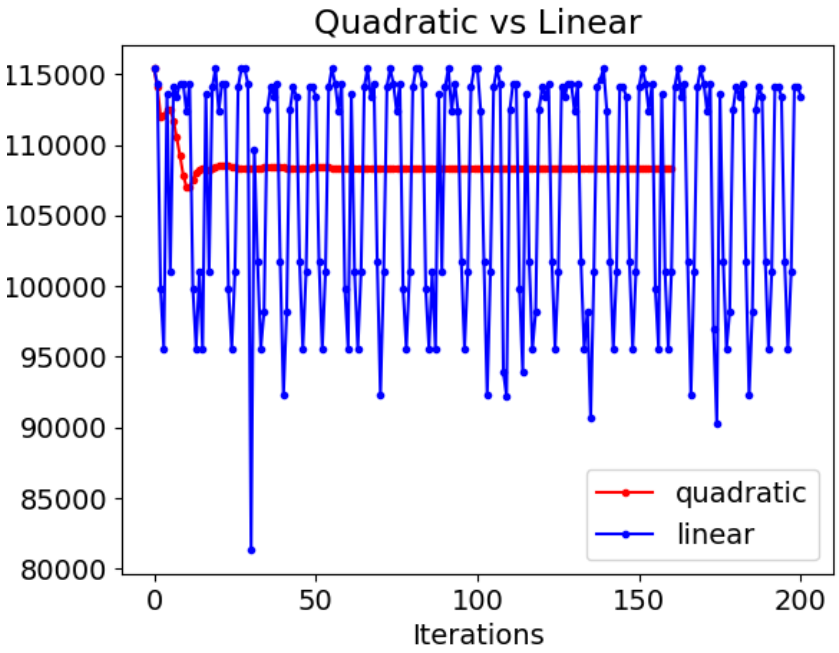


Figure 5.4: Quadratic objective function vs linear objective function

Figure 5.4 shows how the quadratic objective function finished with an optimal objective value of -108390 at iteration 160 but the linear objective function could not find the optimal objective value after it reached 1000(the figure shows until 200) iterations which is the limit number of iterations our implementation allows to find the optimal solution. It found a sub-optimal with an objective value of -112411 at iteration 1000. This figure also shows how the quadratic

formulation, from iteration 20 to iteration 160, is trying to adjust the solution. By contrast, the linear formulation the difference it had from iteration 196 to iteration 200 is enormous repeating the same pattern in the previous and the following iterations.

5.5 Rho policy

In this section we studied the behaviour of the Progressive Hedging applying the rho policies.

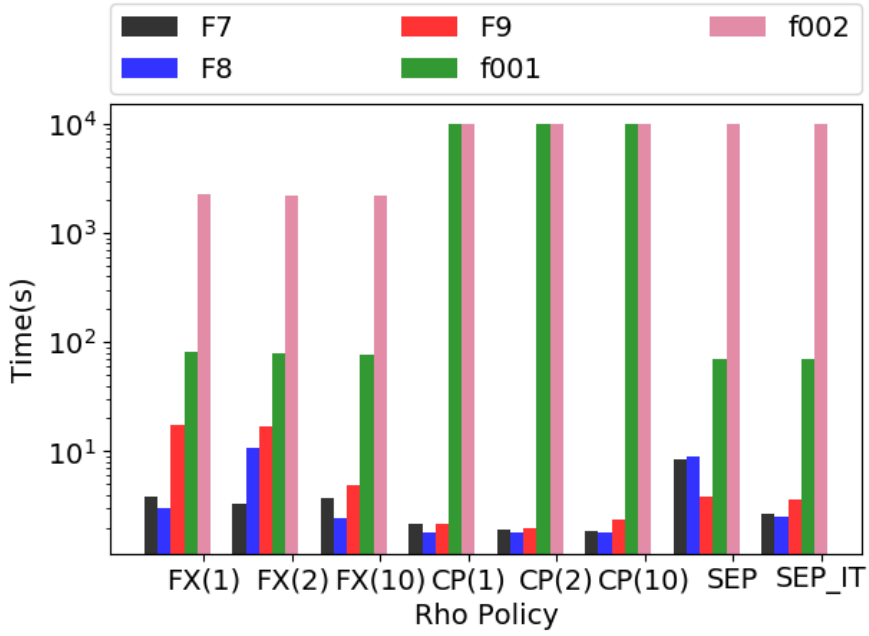


Figure 5.5: Applying the rho policies

Figure 5.5 shows the performance differences between the rho policies. The farmer set instances were solved with good times by all the policies only presenting not good quality objective values the CP policy. Despite the fact that instances f00x contains quite binary variables the solution times increases. Instance f001 could not be solved by CP(*) and only FX(*) could solve f002.

The SEP-IT policy outperformed the others which solved mostly all the instances with the best times. However, it could not solve problem f002 like SEP and CP policies, because these policies are prepared to be run in two-stage stochastic programs so can produce non-convergence if are used in multi-stage stochastic problems.

In instance F9 the policy FX(1) made 88 iterations and found and optimal solution of -111339 which was the same as iteration 34 but at this point the solution of all the scenarios was not considered to be converged because is considered to be equal if the sum of difference of the scenario solutions is minor to $\epsilon = 1^{-6}$. However SEP-IT which was prepared to speed up the convergence in this situations and found the solution at iteration 13.

This figure also shows how sensitive is the Progressive Hedging in terms of setting the good parameters to solve a specific problem. This can cause that a problem which was possible to solve if the adequate parameters were not given then causes that this implementation can not converge.

Chapter 6

Conclusions and Future Work

This work presents how better is our implementation respect pyro-ph and how sensitive it is respect the configuration parameters set to the program. The most significant results show how our implementation reduced significantly the time in comparison to pyro-ph. However for the kind of problems we used our implementation could not beat CPLEX. These results also showed how an adequate update policy can help to improve the research of an optimal solution, how adding the quadratic term improves the research and it is possible that the linear objective function can not find a solution and how influence solving a problem into a specific number of clusters so it can increase the time if it is not solved with the adequate number of clusters.

Depending on the update policy applied to solve a specific problem, it can improve the performance of our algorithm. A study research could be done to develop new policies to update coefficient ρ to improve the performance of solving mixed-integer two-stage problems. The update policies we have applied don't work properly when solving mixed-integer multi-stage problems so new update policies would be needed to make Progressive Hedging converge more quickly. The functionalities after receiving the solutions of the subproblems in the master processor (step 11, step 12 and step 13 of algorithm 1) should be parallelized using threads since the more processors are needed to split large mixed-integer problems into small subproblems, the more workload the master processor have in these operations. Finally, new parallel collaborative designs are required to avoid waiting times and achieve a solution where all the time, all the computational resources are working together and adjusting dynamically the computational resources.

Bibliography

[1] M. S. Bazaraa, J.J Jarvis, and H.D. Sherali. *Linear programming and network flows.*

[2] D. Mahlke. *A scenario tree-based decomposition for solving multistage stochastic programs : with application in energy production.* Vieweg+Teubner Verlag, 2011.

[3] W. A. Chaovalitwongse, K. C. Furman, and P. M. Pardalos. *Optimization and logistics challenges in the enterprise.* Springer-Verlag, 2009.

[4] P. M. Pardalos. *Systems analysis tools for better health care delivery.* Springer, 2013.

[5] H. T. Huynh, V. S. Lai, and I. Soumare. *Stochastic Simulation and Applications in Finance with MATLAB Programs.* John Wiley & Sons, 2011.

[6] C. C. Carøe and J. Tind. L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, 83(1-3):451–464, jan 1998.

[7] L. F. Escudero, M. A. Garín, G. Pérez, and A. Unzueta. Lagrangian Decomposition for large-scale two-stage stochastic mixed 0-1 problems. *TOP*, 20(2):347–374, jul 2012.

[8] J. P. Watson and D. L. Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Comput Manag Sci*, 8:355–370, 2011.

[9] A. Pagès-Bernaus, G. Pérez-Valdés, and A. Tomasgard. A parallelised distributed implementation of a Branch and Fix Coordination algorithm. *European Journal of Operational Research*, 244(1):77–85, jul 2015.

[10] C++ Language: Features, Uses, Applications & Advantages - Hackr Blog.

[11] V. Rajput and A. Katiyar. Proactive Bottleneck Performance Anallysis in parallel computing using openMP. Technical Report 5, 2013.

[12] About OpenNebula – https://opennebula.org/.

[13] About Pyomo http://www.pyomo.org/about.

[14] J. R. Birge and F. Louveaux. *Introduction to stochastic programming.* Springer Series in Operations Research, 1997.

[15] S. Bezuglyi and P. E. T. Jorgensen. Introduction and examples. *Lecture Notes in Mathematics*, 2217:1–12, 2018.